Code-Standard Team Beta

Inhalt

Abbildungsverzeichnis

1. Einführung

ISO 9126
- Änderbarkeit/Wartbarkeit
  - Analysierbarkeit
  - Konformität
  - Modifizierbarkeit
  - Stabilität
  - Testbarkeit
- Benutzbarkeit
  - Attraktivität
  - Bedienbarkeit
  - Erlernbarkeit
  - Konformität
  - Verständlichkeit
- Effizienz
- Funktionalität
- Übertragbarkeit
- Zuverlässigkeit

=> gelb = relevant für Programmierstil

2. Coding Standards JAVA

2. Source File Basis

2.1 File Name
Case-sensitive name of top level class + .java

2.2 File encoding
UTF-8

2.3 Special Characters

2.3.1. Whitespaces characters
Tab characters are not used for indentation!

2.3.2 Special escape sequences
No Unicode escapes
e.g. \b instead of \u000a

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Abbildung 1 - Escaping Characters

2.3.3

Non-ASCII characters

Ok

```
String unitAbbrev = "µs";
```

Not OK

```
String unitAbbrev = "\u03bcs"; // "µs"
```

3. Source File Structure

3.1 License / Copyright information

3.2 Package Statement

3.3 Import Statements

3.3.3 Ordering and spacing

All static imports in single block
All non-static imports in single block

3.4 One top-level class

3.4.2 Ordering of class contents

Logical order instead of chronological order by date added

4. Formatting

4.1 Braces

4.1.1 Optional braces

Used, even if is empty or single statement

4.1.2 Nonempty blocks

Egyptian Brackets
https://blog.codinghorror.com/new-programming-jargon/

OK

```
if ( a == b) {
    printf("Hello");
}
```

NOT OK

```
if (a == b)
{
    printf("Hello");
}
```

4.3 One statement per line

4.5 Line-wrapping

4.5.1 Where to break

No fixed formula

Break at a higher syntactic level

OK

```
MyLambda<String, Long, Object> lambda =
    (String label, ...)
```

NOT OK

```
MyLambda<String, Long, Object> lambda = (String label,
    Long value...)
```

4.5.2 Indent Continuation lines at least +4 spaces

OK

```
MyLambda<String, Long, Object> lambda =
____(String label, ...)
```

NOT OK

```
MyLambda<String, Long, Object> lambda =
_Tab_(String label, Long value...)
```

4.6 Whitespace

4.6.2 Horizontal Whitespace

1. Separating any reserverd word (if, for etc.)

7. Between a double slash

```
( // )
```

8. Between type and variable

```
List<String> list
```

Operators

OK

```
a = (b + c) * d
```

NOT OK

```
a=(b+c)*d
```

Commas

OK

```
fun(a,b,c)
```

NOT OK

fun(a , b, c)

4.6.3 Horizontal alignment

```
private int x; // this is fine
private Color color; // this too

private int   x;      // permitted, but future edits
private Color color;  // may leave it unaligned
```

Abbildung 2 - Horizontal Alignement

4.8 Specific constructs

4.8.2 Variable Declarations

4.8.2.1. One Variable per declaration

OK

int a;
int b;

NOT OK

int a,b;

4.8.2.2 Declared when needed

Declared when first used

4.8.4 Switch statements

4.8.4.1 Indentation

+2 blanks

4.8.4.2 Comments

Either

Break / continue / return Statement
Comment

```
switch (input) {
  case 1:
  case 2:
    prepareOneOrTwo();
    // fall through
  case 3:
    handleOneTwoOrThree();
    break;
  default:
    handleLargeNumber(input);
}
```

Abbildung 3 - Comments

4.8.4.3 Presence of default label

4.8.5 Annotations

4.8.5.1 Type-use annotations

Directly before annotated type

final @Nullable String name;

4.8.5.2 Class annotations

One annotation per line

4.8.6. Comments

4.8.6.1.

```
/*
 * This is          // And so           /* Or you can
 * okay.            // is this.          * even do this. */
 */
```

Abbildung 4 - Allowed Comments

5. Naming

5.1 Rules common to all identifiers

Only ASCII letters and digits

NOT

name_, mName, s_name, kName

5.2. Rules by identifier type

5.2.1 Package names

OK

com.example.deepspace

NOT OK

com.example.deepSpace

5.2.2 Class names

Class names

UpperCamelCase

Test

Ends with Test

UpperCamelCaseTest

5.2.3 Method names

Methods

lowerCamelCase

Verb phrases

Test
    JUnit Tests

5.2.4 Constant Names
UPPER_SNAKE_CASE
    static fields

5.2.5 Non-constant field names
lowerCamelCase

5.2.6 Parameter Names
lowerCamelCase

5.2.7 Local variable names
lowerCamelCase

5.2.8 Type variable names
Single Capital Letter
    E, T, T2
Name with capital T
    RequestT

5.3 Camel Case: defined
"XML HTTP request"
    OK
        XmlHttpRequest
    NOT OK
        XMLHTTPRequest

6. Programming Practices

6.2 Caught Exceptions: not ignored
At least comment when no action is taken
    OK
```
        try {
                int i = Integer.parseInt(response);
               return handleNumericResponse(i);
        } catch (NumberFormatException ok) {
          // it's not numeric; that's fine, just continue
        }
```

7. Javadoc

```
/**
* Diese Funktion macht xy
*/
public bla bla bla
```

Quellen

https://google.github.io/styleguide/javaguide.html

https://www.geeksforgeeks.org/coding-guidelines-in-java/

https://de.wikipedia.org/wiki/ISO/IEC_9126